Assignment 4

Team member:
Bo Li,
ID: 913020815

Jiawen Zhu
ID: 916561820

JDK version: 1.8

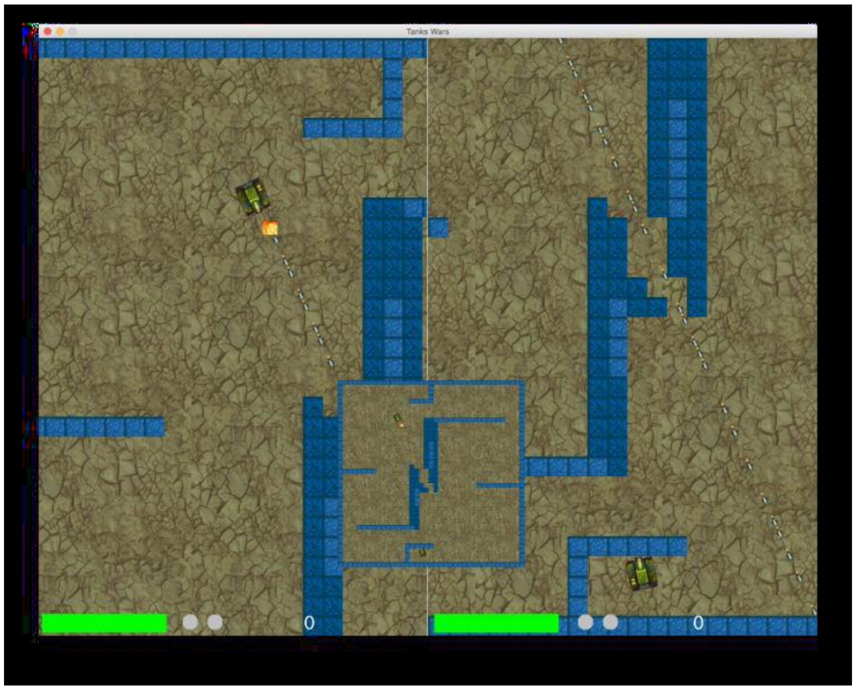
5/23/2017

Github: https://github.com/SFSU-CSC-413/assignment-4-bo-jiawen-team.git

1. Introduction/Overview:

In both Tank game and Galactic Mail game, our main focus is to practice java GUI. To do that, we use MVC (model view control) as the design pattern.

In the tank game, the model includes (Animation, BulletAnimation, GameObject, TankObject, WallObject, Player, Sound and Sprite) classes. **The view includes** (AnimationPanel) which will display game objects on the panel and (GameFrame) which will display panel on the Frame. **Controller includes** (KeyboardPanel) which takes the user input and translates to the object's motion in the game. Addition, (KeyboardAnimation) is the controller since it is the main class.



In the Galactic Mail game, the model includes (Animation, AsteroidAnimation, MoonAnimation, GameObject, ShipObject, Sound and Sprite) classes. **The view includes** (AnimationPanel and GameFrame). **Controller includes** (KeyboardPanel and KeyboardAnimation). Many classes are **reused**; they are (Animation, AnimationPanel, KeyboardPanel, GameObject, Sprite and Sound).

Classes' description:

Used in both Games:

Game Frame:
a. Creates an instance of a AnimationPanel which extends JPanel.

Animation:
a. Creates an animation

Sprite:
a. Load a file image and store in an array.
b. Uses getFrame to get a single image based on the index passed in.

KeyBoardAnimation:
a. Shows the tank movements when a single key or multiple keys are being pressed.
b. Explosion will happen when a tank is fire.

GameObject:

a. Loads an image file.
b. Sets the location of an object.
c. Gets the location of an object.

d. Draw the image based on its location.

Tank Game:

AnimationPanel:

a. Creates two instances of tank and player objects.
b. Draw mini map
c. Draw partial map based on the current tank movement.
d. Repaint the health, the lives and points for each player

BulletAnimation:

a. Extends Animation.
b. Sets the speed and direction of a bullet.
c. Detects where bullet collide with the wall.
d. Detects where bullet collide with tank.
e. Repaints the movement of the bullet.
f. Stops the bullets when collision occur.

TankObject:

a. Calculates the rotating angels.
b. Passes the angel to Spirit class and picks a specific image.
c. Rotates the tank direction based on the angles.
d. move tank forward.
e. move tank backward.
f. Sets the open fire position.

WallObject:

a. Loads the image of the wall
b. Repaint the image of the wall

Player:

a. Creates a tank instance
b. Initialize the health, lives and points for the player.
c. Deduct and Earn points.
d. Repaint the health, lives and points.

Galactic Mail Game:

KeyboardPanel:

a. Implementing keyListener.
b. Handel the action when user presses keys.

AnimationPanel:

a. Repaint the asteroids.
b. Repaint the moons.

c. Calculate the score and display on current panel.

AsteroidAnimation:

a. Extends from Animation.
b. Create one AsteroidAnimation object.
c. Sets it's angles and directions (x,y).
d. Repaint the current Asteroid.

MoonAnimation:

a. Extends from Animation.
b. Creates one MoonAniation object.
c. Set it's angles and directions(x,y).
d. Repaint the current moon.

ShipObject:

a. Extends from GameObject
b. Sets rotation
c. Sets current and next location
d. Sets boundary
e. Repaint the ship.

Important methods in the tank game:

a. Methods in TankObject class:

i. Collision with wall method:

```
public boolean collisionToWall(WallObject wall){…}
```

Run through wall array and detects where tank's rectangle intersects with wall's rectangle. Tank will stop to move where the two rectangle interest.

ii. Collision with tank method:

```
public boolean collisionToTank(TankObject anotherTank){…}
```

Where one tank's rectangle intersects with another tank's rectangle.

iii. Move methods:

The total numbers of the images are 60 degree and total angle are 360 degrees, so each angle is 6 degree before converting it to radians. The index starts from 1 to 60 based on the rotating steps.

```
public void forwardMove() {
   this.nextX = this.getX() + (int) (Math.cos(Math.toRadians(angle[index])) * STEP);
   this.nextY = this.getY() - (int) (Math.sin(Math.toRadians(angle[index])) * STEP);
}
```

Using Trigonometric (sin and cos) to calculate the direction when tank is moving forward.

Similar to forward Move. Using Trigonometric (sin and cos) to calculate the direction when the tank is moving backward.

iv. initial shooting position:

fireStartX:
return the position when tank starts fire in x direction

fireStartY:
return the position when tank starts fire in y direction

v. Rotating method:
Using a method getFrame in the Sprite class to get specific image based on the index it passed in.

b. Methods in WallObject class:

i. setsArray:
Read given wall file(.txt) and assign it a wallArray(works like lexer). Reads 1 as breakable wall and 2 as unbreakable wall.

ii. setsNewArray: When a collision happens, a new wall array will be created. This method will create a new array for the wall object.

repaint:
Repaint the breakable and unbreakable wall based on the “2” and “1”.

c. Methods in BulletAnimation class

i. **public boolean** collisionToWall() {…}
Return true when a bullet hit a breakable wall.

ii. **public void** attackWall() {…}
Reassign the value in the wall array.

iii. **public boolean** collisionToTank() {…}
Return true when a bullet hit a Tank.

iv. **public boolean** collisionToTank() {…}
The tank will be damage and the player will lose the blood.

v. **public void** explosion(TankObject diedTank) {…}
Using Animation class to get explostion animation.

vi. **public void** forwardMove() {..}
Relocate the bullet moving motion based on the speed.

vii. **public void** repaint(Graphics graphics) {…}
Repaint the bullet animation and the walls when walls are being damaged.

d. Methods in Player class:

i. **public void** paintHealth(Graphics graphics, **int** x, **int** y) {…}
This method will draw the health bar which includes three colors on the panel. The colors are green, yellow and red.

ii. **public void** paintLives(Graphics graphics, **int** x, **int** y) {…}
This method shows how many lifes a player has. There light gray circles indicate three lifes.

iii. **public void** paintPoint(Graphics graphics, **int** x, **int** y) {…}
This method shows how many points a player earn after each time another player terminate.

e. Methods in KeyBoardAnimtion:

```
private boolean upPress, downPress, leftPress, rightPress, enterPress;
private boolean wPress, sPress, aPress, dPress, spacePress;

@Override
public void keyPressed( KeyEvent e ) {…}

@Override
public void keyReleased( KeyEvent e ) {…}

@Override
public void run() {…}
```

When keys are being pressed or released, it's going to create one single Boolean variable or several Boolean variables. When the variable is true, it will invoke the run method. The run method will update current location or image for the ship.

Finally, the repaint method will update ship image display on the panel.

Important methods in the Galactic Mail Game

a. Methods in ShipObject class:

A lot of methods are copied from TankObject class. Such as rotatedLeft(), rotatedRight(). The SetDegree method in both Galactic Mail and in Tank Game are identical. The degree increases in Tank game is 6, because there are 60 images and each image is 6 degrees; The degree increases in the Galactic Mail game is 5, since there are 72 images and each image is 5 degrees.

b. Methods in Animation class:

i.
```java
public void move() {
    this.x += (int) (Math.cos(Math.toRadians(angle[randomDegree])) * speed);
    this.y += (int) (Math.sin(Math.toRadians(angle[randomDegree])) * speed);
}
```

Move method are being called in both AsteroidAnimation and MoonAnimation to relocated both asteroid and moon.

ii. `public void setInitposition() {…}`
This method sets an initial random position.

iii. `public void outOfBound() {…}`
This method sets a boundary for the both asteroid and moon. When they move out of boundary, they will appear on the opposite side of the screen. For example, if the moon leaves from an east of screen, the same moon will come back from west of the screen.

c. Methods in AnimationPanel class:

i.
```java
 @Override
public void paintComponent( Graphics graphics ) {…}
```

Starting a game. Sets 4 number of moons and 4 number of asteroids.

Checking to see if a ship is terminated. Using the die Boolean to check if the ship is terminated in the current game. When the ship is terminated, the score will be displayed on the panel.

Calculate the score based on the how many time the ship land on the moon. When the ship is landed on the moon, get 1000+ points. If the ship is still on the moon after the ship gets point, the ship will lose points. This rule is based on the google search result.

Setting the coordinates and display on the screen.
When a ship is landed on the moon, the coordinates of the ship is same as coordinates of the moon.

Display top 5 scores after the ship are terminated.

ii. `public void highScore() {…}`

Store the scores in the ArrayList of the score. Using bubble sort to sort the ArrayList and save top 5 score in the ArrayList.

2. Command line instructions to compile and execute:

For the tank game: Open TextEdit and copy this line below
Main-Class: TankGame.application.examples.KeyboardAnimation
Locate your src folder on terminal
The put this commond line on your terminal

```
jar cvfm GalacticMail.jar GalacticMailManifest.txt .
```

For the GalacticMail game: Open another TextEdit and copy this line below
Main-Class: GalacticMail.application.testing.KeyboardAnimation
Locate your src folder on terminal
The put this commond line on your terminal

```
jar cvfm GalacticMail.jar GalacticMailManifest.txt .
```

3. Assumptions:

Our tasks for each member are different. Assume one member is working on one task without using another member's code. One person needs to know what one method will return before another person even write the first line of that method, so team member needs to communicate to decide what method signatures are and what method will return.

Example1: In the TankObject class. The method collisionToWall has an object as its signature right now, but before I (Jiawen) worked on this method, I didn't know whether this method will pass in an object or 2-D array. So, I assume this method will pass an array. After merging my code with Bo's code, we came up our own decision that a wall is also an object. I changed it to Wall object in the collision method later.

Example2, Initially, Bo and I assume we should have bullets class. But after creating a BulletAnimation class, we decide that a bullet is only a **rectangle object**.

```
Rectangle bullets = new Rectangle(x, y, 10, 10);
```

It's x and y are based on the tank object x and y positions.

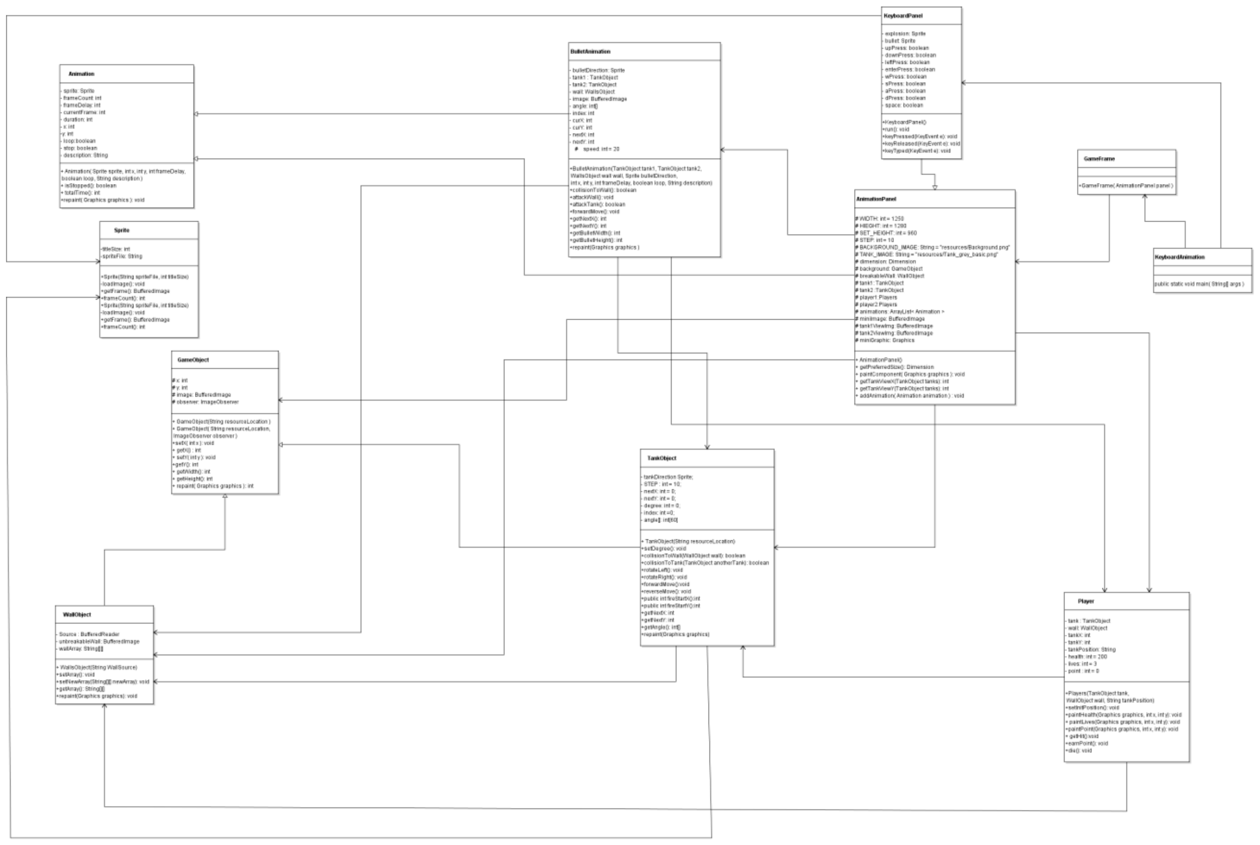
In the Galactic Mail game:

The ship images are created in the ShipObject class. It should be put into the GameObject to reduce a problem when adding more ship in the game, However, we think it will be OK to put the image in the ShipObject for now since we only have one ship object in this game.

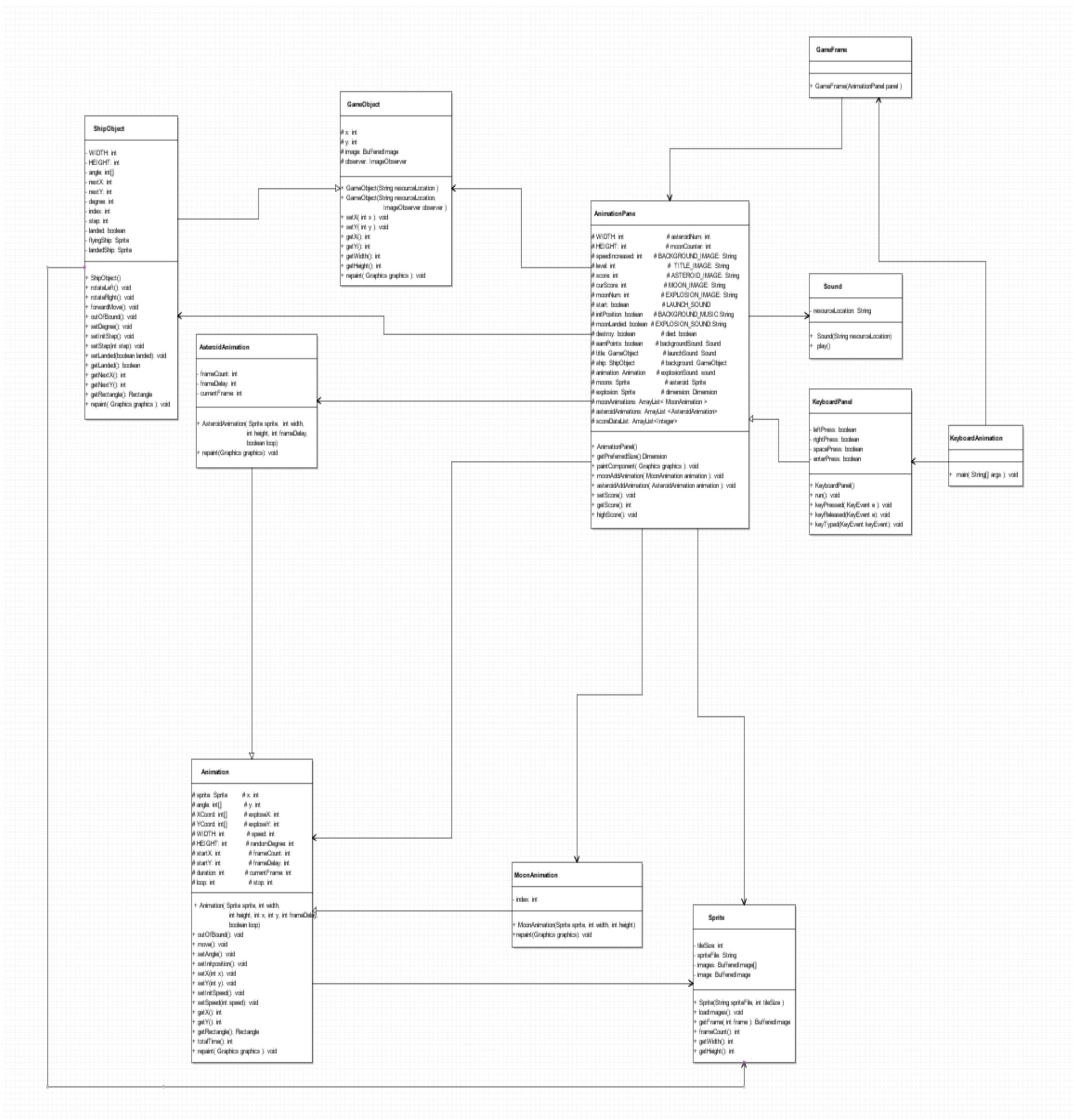
We didn't use observable to update each character in the game. Instead, we use

4. Implementation discussion

UML class diagram for Tank game:



UML diagram for the Galactic Mail game


GameFrame
+ GameFrame(AnimationPanel panel)
GameObject
# x: int
# y: int
# image: BufferedImage
# observer: ImageObserver
+ GameObject(String resourceLocation)
+ GameObject(String resourceLocation, ImageObserver observer)
+ setX(int x): void
+ setY(int y): void
+ getX(): int
+ getY(): int
+ getWidth(): int
+ getHeight(): int
+ repaint(Graphics graphics): void
ShipObject
- WIDTH: int
- HEIGHT: int
- angle: int[]
- nextX: int
- nextY: int
- degree: int
- index: int
- step: int
- landed: boolean
- flyingShip: Sprite
- landedShip: Sprite
+ ShipObject()
+ rotateLeft(): void
+ rotateRight(): void
+ forwardMove(): void
+ outOfBound(): void
+ setDegree(): void
+ setInitStep(): void
+ setStep(int step): void
+ setLanded(boolean landed): void
+ getLanded(): boolean
+ getNextX(): int
+ getNextY(): int
+ getRectangle(): Rectangle
+ repaint(Graphics graphics): void
AnimationPane
# WIDTH: int
# asteroidNum: int
# HEIGHT: int
# moonCounter: int
# speedIncreased: int
# BACKGROUND_IMAGE: String
# level: int
# TITLE_IMAGE: String
# score: int
# ASTEROID_IMAGE: String
# curScore: int
# MOON_IMAGE: String
# moonNum: int
# EXPLOSION_IMAGE: String
# start: boolean
# LAUNCH_SOUND
# initPosition: boolean
# BACKGROUND_MUSIC:String
# moonLanded: boolean
# EXPLOSION_SOUND:String
# destroy: boolean
# died: boolean
# earnPoints: boolean
# backgroundSound: Sound
# title: GameObject
# launchSound: Sound
# ship: ShipObject
# background: GameObject
# animation: Animation
# explosionSound: sound
# moons: Sprite
# asteroid: Sprite
# explosion: Sprite
# dimension: Dimension
# moonAnimations: ArrayList< MoonAnimation >
# asteroidAnimations: ArrayList <AsteroidAnimation>
# scoreDataList: ArrayList<Integer>
+ AnimationPanel()
+ getPreferredSize():Dimension
+ paintComponent(Graphics graphics): void
+ moonAddAnimation(MoonAnimation animation): void
+ asteroidAddAnimation(AsteroidAnimation animation): void
+ setScore(): void
+ getScore(): int
+ highScore(): void
Sound
- resourceLocation: String
+ Sound(String resourceLocation)
+ play()
KeyboardPanel
- leftPress: boolean
- rightPress: boolean
- spacePress: boolean
- enterPress: boolean
+ KeyboardPanel()
+ run(): void
+ keyPressed(KeyEvent e): void
+ keyReleased(KeyEvent e): void
+ keyTyped(KeyEvent keyEvent): void
KeyboardAnimation
+ main(String[] args): void
AsteroidAnimation
- frameCount: int
- frameDelay: int
- currentFrame: int
+ AsteroidAnimation(Sprite sprite, int width, int height, int frameDelay, boolean loop)
+ repaint(Graphics graphics): void
Animation
# sprite: Sprite
# x: int
# angle: int[]
# y: int
# XCoord: int[]
# exploseX: int
# YCoord: int[]
# exploseY: int
# WIDTH: int
# speed: int
# HEIGHT: int
# randomDegree: int
# startX: int
# frameCount: int
# startY: int
# frameDelay: int
# duration: int
# currentFrame: int
# loop: int
# stop: int
+ Animation(Sprite sprite, int width, int height, int x, int y, int frameDelay, boolean loop)
+ outOfBound(): void
+ move(): void
+ setAngle(): void
+ setInitposition(): void
+ setX(int x): void
+ setY(int y): void
+ setInitSpeed(): void
+ setSpeed(int speed): void
+ getX(): int
+ getY(): int
+ getRectangle(): Rectangle
+ totalTime(): int
+ repaint(Graphics graphics): void
MoonAnimation
- index: int
+ MoonAnimation(Sprite sprite, int width, int height)
+ repaint(Graphics graphics): void
Sprite
- tileSize: int
- spriteFile: String
- images: BufferedImage[]
- image: BufferedImage
+ Sprite(String spriteFile, int tileSize)
+ loadImages(): void
+ getFrame(int frame): BufferedImage
+ frameCount(): int
+ getWidth(): int
+ getHeight(): int

5. Results and conclusions:

What we learn from this project is huge. We have learned how to define every single game object(characters) in terms of OOP. We have created the TankObject and WallObject in the Tank game; ShipObject in the GalacticMail game. We also learn how to create an animation using overridden repaint method.

We also use MVC as our design pattern. By using MVC, we are able to make our code cleaner and easier to read for future improve.
Both two games are working fine, but our implementations are not as good as what we want it to be. If we have more time, we would like to improve the following:

Loose coupling

When a user wants to add a third player in the game, we should have only changed the KeyBoardPanel which is the controller. If we have more time, we would decrease the coupling between Player class and any other classes rather than the controller.

Code clean

Bo and I worked very hard to fill the requirement implementing the methods in the game. We had a little time to go back and work on the code organization. Some of the code are redundant. For instance, there are two methods that will get rid of the wall. One is checking if the wall is there, another one is to delete the breakable wall image. collisionToWall method run through the 2-D wall array to check where collision happens, but attackWall run through the same array again. We could reduce the number of code by calling the collisionToWall method in the attackWall method.

Resources files issue:

We found a problem that the resource package should be placed into src file, then the jar file could be created. However, as the IDE we used, the resources package should be out of src file can be run, but the one we submitted to the Github is not in the correct location. This problem we found out when we were doing documentation.

We already give the instruction on documentation on that problem, but we want to make grader clone the project and open it with easy way as the convenience, we just re-push the project, which means we place the resources package into src file in Github Because I want the grader just to open it easier to create jar file and run the project efficiency. And we won't change any piece of code. **My team has told John Roberts and he said it was OK to resubmit this on the GitHub. It won't count as the late submission.**